UML Diagrams

Shawn R Smith

CST499 Software Technology Capstone

University of Arizona Global Campus

Dr. Butler

March 4, 2024

UML MODELS

Figure 1

Class Diagram

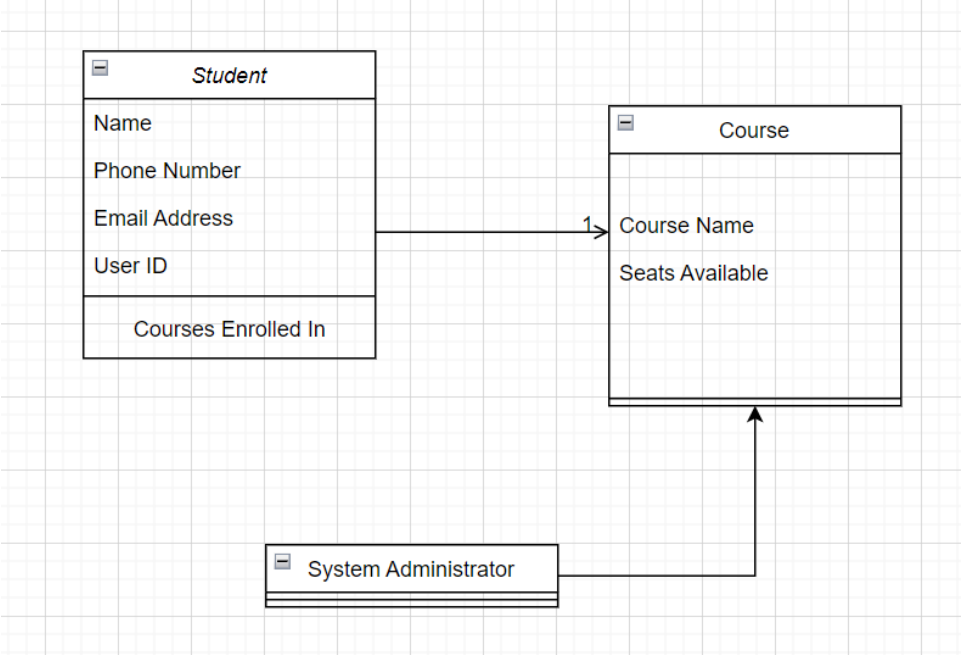


Figure 2

Sequence Diagram

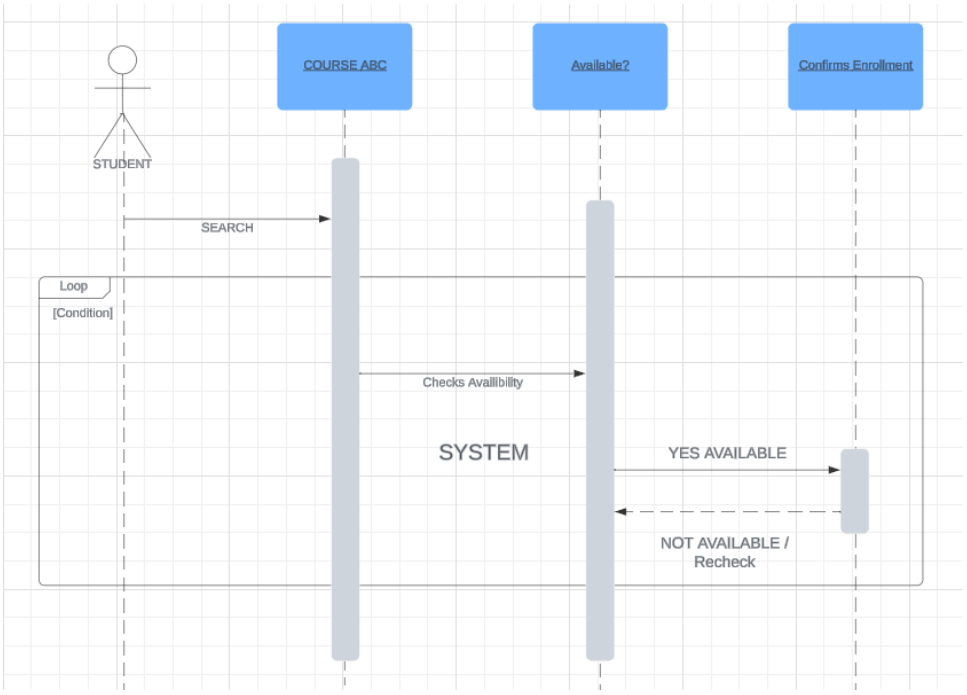
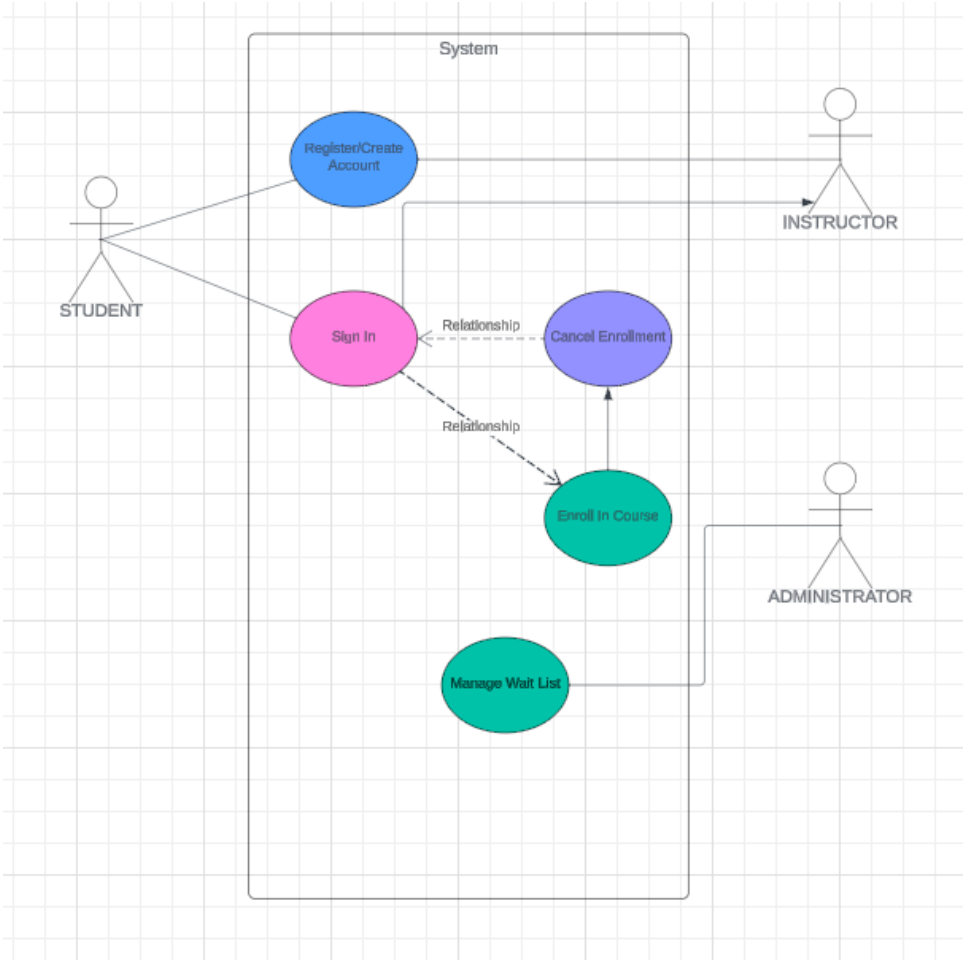


Figure 3

Use Case Diagram



Software systems today have become increasingly complex puzzles, consisting of many interlocking pieces working harmoniously. Ensuring high quality of software is vital in today's marketplace and testing plays a pivotal role in software quality. There are a number of well established, hierarchical levels of testing that systematically verify critical software system components and their validity against user requirements. Component testing uncover module defects, integration testing exposes interaction faults and system testing catches oversights or

challenges in end-to-end workflows. Lastly, final acceptance gives the stamp of approval that the solution or system addresses the agreed upon business needs.

Component Testing

Component testing focuses on verifying the functionality of individual components in isolation.

- Tests are conducted by developers on single modules, classes, functions, or libraries to validate unit functionality and logic flow. Helps identify bugs early before propagation.
- Typical component tests would include API testing, feature testing, unit tests, stability testing and error handling. Test inputs, outputs, APIs, return codes and performance.
- Test environments simulate dependencies that components need with stubs and test harnesses. All tests automated for regression testing after fixes.
- Critical for developers to understand component thoroughly - features, interactions and pain points. Enables comprehensive testing coverage.

Integration Testing

Integration testing verifies combined components still function correctly together to achieve larger functions.

- Modules previously tested and verified during component testing are combined into subsystems for further validation as a group.

- Testing of the communication between components, interface defects, sync issues, system-level defects.
- Performed top-down or bottom-up. Stub and drivers are used to simulate unavailable modules for progressive testing.
- Includes pattern-based tests, interaction tests, environment verification, integration usability and user acceptance.

System Testing

System testing checks conformance to requirements for the entire integrated software product before release.

- Uses real data, configurations, software/hardware for thorough testing of overall system features, security, performance.
- Test cases derived from use cases, system requirements and behaviors documented.
- Alpha and beta testing done with both internal members as well as external user representatives if applicable.
- Covers recovery testing, load testing, volume testing, downgrade testing and platform change testing etc.

Acceptance Testing

Formal sign-off process for end user/customer to officially accept/reject system based on verification of requirements being met.

- Carried out in production-like environment using real-life data sets. Utilizes real user scenarios and use cases.
- Variants include user acceptance testing (UAT), operational acceptance testing (OAT), business acceptance testing (BAT).
- Contracts/agreements may require an acceptance certificate from customers before payment or implementation approval.

Well-structured testing methodologies have proven to reduce project risk through progressive inspection of the evolving product - steadily gaining confidence in components before combining them into bigger subgroups and eventually the final system. Component tests establish foundation, integration tests confirm harmonious data and control flow between cooperating units, system tests validate the production system meets specifications while acceptance testing provides the official customer validation that the delivered system solves the actual business problem. Using this tiered testing strategy ensures thorough assessment and verification of quality for enterprise software solutions.

Reference:

Rao, K. N., & Sastri, Y. (2015). Software testing and quality assurance: Theory and practice.

Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing.

Sommerville, I. (2011). Software engineering.

Tsui, F., Karam, O., & Bernal, B. (2018). *Essentials of software engineering* (4th ed.)